Proactive Algorithms for Scheduling with Probabilistic Durations*

J. Christopher Beck
Department of Mechanical & Industrial Engineering
University of Toronto, Canada
jcb@mie.utoronto.ca

Nic Wilson
Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland
n.wilson@4c.ucc.ie

Abstract

Proactive scheduling seeks to generate high quality solutions despite execution time uncertainty. Building on work in [Beck and Wilson, 2004], we conduct an empirical study of a number of algorithms for the job shop scheduling problem with probabilistic durations. The main contributions of this paper are: the introduction and empirical analysis of a novel constraint-based search technique that can be applied beyond probabilistic scheduling problems, the introduction and empirical analysis of a number of deterministic filtering algorithms for probabilistic job shop scheduling, and the identification of a number of problem characteristics that contribute to algorithm performance.

1 Introduction

Classical models of scheduling assume all information is known and static. When faced with uncertainty, *proactive* scheduling techniques seek to incorporate models of the uncertainty in the off-line problem solving and to build solutions that can achieve a satisfactory level of performance at execution time. [Beck and Wilson, 2004] addressed the problem of job shop scheduling with probabilistic durations with two styles of algorithm both using Monte Carlo simulation to evaluate the probabilistic makespan of solutions. This paper builds on that work, making the following contributions: (i) A novel constraint-based search technique is introduced that performs a series of complete branch-and-bound searches on highly constrained problem models. The models are gradually relaxed with the cost of the best solution model found in the previous searches being used as an upper bound. Experimental results show that the algorithm performs as well as the existing branch-and-bound style of search on small problems and significantly out-performs it on larger problems. (ii) A number of novel deterministic filtering algorithms are presented. On larger problems such algorithms out-perform the other algorithms tested. Their performance is affected by the following two factors: the quality of the deterministic solutions found and the correlation between the deterministic and probabilistic makespan of solutions. (iii) It is demonstrated that the method introduced by Beck & Wilson for the incorporation of uncertainty in deterministic problems can increase the correlation between deterministic and probabilistic makespan, providing an explanation for its strong performance.

In the next section we briefly review previous work. The six search algorithms investigated in this paper are defined in Section 3 and Section 4 presents our experiments and results. In Section 5, we present our conclusions.

2 Background

The *job shop scheduling problem* (JSP) consists of a set $\mathcal{A}$ of activities each with a positive integer-valued duration, d_i. $\mathcal{A}$ is partitioned into *jobs*, and with each job is associated a total ordering on that set of activities. Each activity specifies a resource on which it must execute without interruption. No activities that require the same resource can overlap in their execution. We represent this formally by a partition of $\mathcal{A}$ into *resource sets*. A *solution* consists of a total ordering on each resource set such that the union of the resource and job orderings is an acyclic relation on $\mathcal{A}$. The *makespan* of a solution is the difference between the minimum start time and the maximum end time of all activities. Finding a solution with minimum makespan is NP-hard [Garey and Johnson, 1979]. An *independent probabilistic job shop scheduling problem* is a JSP where the duration $\mathbf{d}_i$ associated with an activity $A_i \in \mathcal{A}$ is a positive integer-valued random variable. These random variables are fully independent. $\mathbf{d}_i$ has expected value $\mu_i = \mathrm{E}[\mathbf{d}_i]$ and variance $\sigma_i^2 = \mathrm{Var}[\mathbf{d}_i]$. The makespan $\mathbf{make}(s)$ of solution s is therefore also a random variable. We fix a degree of confidence p, e.g., we set $p = 0.95$ in the experiments. The minimum probabilistic makespan, $D(s)$, of a solution s is the smallest value D such that the probability that all jobs will finish before D is at least p. That is, $D(s) = \min\{D : \Pr(\mathbf{make}(s) \leq D) \geq p\}$. An optimal probabilistic makespan is the minimum $D(s)$ over all solutions s.

Beck & Wilson introduce an analytical model and develop two main results. First, it is demonstrated that Monte Carlo simulation can be used to find the probabilistic makespan for a solution and to find a lower bound on the probabilistic makespan of a partial solution. Second, it is shown that a deterministic JSP instance can be constructed from a probabilis-

*This material is based upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075 and by ILOG, SA.

tic JSP instance by setting the duration d_i of each activity to $\mu_i + q\sigma_i$ and that, for the appropriate choice of non-negative q, the optimal makespan of the deterministic instance is a lower bound on the optimal probabilistic makespan of the corresponding probabilistic JSP. Three algorithms are defined: a branch-and-bound approach where Monte Carlo simulation is used at each node to find a lower bound (at internal nodes) or an upper bound (at a leaf node) on the probabilistic makespan, and two deterministic filtering algorithm consisting of using either constraint-based search or tabu search to find a series of increasingly good deterministic solutions, each of which is simulated. Empirical results indicated that the first algorithm was able to find optimal solutions only for very small problems, that for medium-sized problems the constraint-based filtering algorithm found the best solutions, and for the largest problems the tabu-based filtering algorithm performed best. Choosing $q > 0$ led to stronger performance of the filtering algorithms. No empirically founded explanations were provided to explain the differences in problem solving performance.

3 Algorithms for Probabilistic JSP

In this section, we describe six algorithms for solving the probabilistic job shop scheduling problem.

3.1 Approximately Complete Branch-and-Bound

We implemented a branch-and-bound algorithm identical to that of Beck & Wilson except that we use stronger heuristics that make decisions on the sequence of activities in each resource set (see Section 3.5). We refer to this algorithm as the *B&B-N* as it performs *B*ranch-and-*B*ound with simulation at each *N*ode.

3.2 Iterative Descending Bound Search

By setting activity durations based on a q value that ensures a lower bound on the probabilistic makespan, we can use standard constraint propagation on the deterministic durations to implicitly calculate a lower bound at each internal node. At each leaf node, simulation is used as in B&B-N to find the probabilistic makespan. Rather than parameterize the algorithm with a fixed q value, we perform repeated tree searches with a descending q value. Starting with a high q value, we begin a tree search. When the first leaf, e, is reached, simulation is used to find $D(s_e)$. It is likely that $D_q(s_e)$, the minimum makespan of s_e in the corresponding deterministic problem, will be larger than $D(s_e)$. Since we enforce $D_q(s_e) \leq D(s_e)$, estimating $D(s_e)$ through simulation causes the search to backtrack to a interior node, i, very high in the tree. At node i, $D_q(S_i) \leq D(s_e)$: S_i is the set of solutions in the subtree below node i and $D_q(S_i)$ is a lower bound on the deterministic makespan of those solutions based on standard constraint propagation. With high q values, we commonly observe that there are very few nodes that meet this criterion and the search space is quickly exhausted. We then decrement the q value (e.g., by 0.05) and start a new tree search. Eventually, and often very quickly, we reach a q value such that there exists a full solution where $D_q(s_e) \leq D(s_e)$. The probabilistic makespan of the best solution that has been found by any point in the overall search is used, as in B&B-N, as an upper bound on subsequent search.

B&B-DQ-L():
Returns the solution with lowest probabilistic makespan

1 $(s^*, D^*) \leftarrow$ findFirstB&BSimLeaves(∞, 0)
2 $q \leftarrow q_0$
3 **while** $q \geq 0$ *AND not timed-out* **do**
4 (s, D) $\leftarrow$ findOptB&BSimLeaves(D^*, q)
5 **if** $s \neq NIL$ **then**
6 $s^* \leftarrow s$; $D^* \leftarrow D$
end
7 $q \leftarrow q - q_{dec}$
end
8 return s^*

Algorithm 1: B&B-DQ-L

Algorithm 1 presents pseudocode for the basic algorithm. We make use of two functions not defined using pseudocode: *findFirstB&BSimLeaves(c, q)* creates a JSP with activity durations defined based on the q value and conducts a branch-and-bound search where Monte Carlo simulation is used for each leaf node and standard constraint propagation is used at interior nodes. The first solution that is found whose probabilistic makespan is less than c is returned. *findOptB&BSimLeaves(c, q)* does the same as findFirstB&BSimLeaves(c, q) except the solution with lowest probabilistic makespan is returned rather than the first one. If no solution is found, a NIL value is returned. Unless the q value is low enough that the deterministic makespan is a lower bound on the probabilistic makespan, this function does not necessarily return the globally optimal solution. We find a starting solution with $q = 0$ to serve as an initial upper bound on the probabilistic makespan. In practice, B&B-DQ-L is run with a maximum CPU time. If $q = 0$ is reached within the time limit, this algorithm is approximately complete.

We refer to this algorithm as *B&B-DQ-L* as it does a series of *B*ranch-and-*B*ound searches with *D*escending *Q* values and with simulation is used at the *L*eaves of the tree.

3.3 Branch-and-Bound Filtering Algorithms

Preliminary experiments with the branch-and-bound filtering algorithm presented by Beck & Wilson showed that a significant amount of simulation was being done early in the search when the upper bound on the deterministic makespan was relatively high. Such solutions would seem to have little chance of being the optimal probabilistic solution as the deterministic solution is very poor. We present two constraint-based filtering algorithms that seek to avoid these wasted simulations.

Branch-and-Bound Timed Better Solution A simple method is to spend a fixed amount of CPU time, $t_{initial}$ to find a solution, s^*, with a low deterministic makespan $D_{initial}$ using a fixed q value and constraint-based search. Then search can be restarted using the same q value and using $D_{initial}$ as an upper bound on the deterministic makespan. Whenever a solution, s, is found such that $D_q(s) \leq D_{initial}$, a simulation is run to evaluate the probabilistic makespan. The solution

B&B-TBS(q):
Returns the solution with lowest probabilistic makespan

```
1  (s*, D_initial) ← findOptB&B(∞, q, t_initial)
2  D* ← ∞
3  while termination criteria unmet AND not timed-out do
4      (s, D) ← findNextB&B(D_initial, q, t_remaining)
5      D' ← simulate(s)
6      if D' < D* then
7          s* ← s; D* ← D'
       end
   end
8  return s*
```

Algorithm 2: B&B-TBS

with the lowest probabilistic makespan is returned when either the entire tree has been searched or when the maximum allowed CPU time has expired. Algorithm 2 contains pseudocode for this approach.

As above, we make use of a number of functions not defined with pseudocode: *findOptB&B(c, q, t)* creates a JSP with activity durations defined based on q and conducts a deterministic branch-and-bound search for up to t CPU seconds using c as an upper bound on the makespan. When the search tree is exhausted or the time-limit is reached, the best solution found, together with its makespan, are returned. No simulation is done. *findNextB&B(c, q, t)* produces a sequence of solutions (one solution each time it is called) whose deterministic makespan is less than or equal to c. The problem is defined using the q value and t is the CPU time limit. The solutions produced are the leaves of the B&B search tree in the order encountered by the algorithm. The c value does not change. Given enough CPU time, the algorithm will evaluate the probabilistic makespan of all solutions whose deterministic makespan is less than or equal to $D_{initial}$. Finally, *simulate(s)* runs Monte Carlo simulation on solution s and the probabilistic makespan is returned.

The algorithm is called *B&B-TBS* for *B*ranch-and-*B*ound *T*imed *B*etter *S*olution. The algorithm is not complete, as there is no guarantee that the optimal probabilistic solution will have a deterministic makespan less than $D_{initial}$.

Branch-and-Bound Iterative Best Solution A more extreme filtering algorithm first finds an optimal deterministic solution and uses it as a filter for choosing the solutions to simulate. Using a fixed q value, a solution with an optimal deterministic makespan, D_q^*, is found and simulated. Then we execute a series of iterations beginning with $i = 0$. For each iteration, all solutions, s_e, with deterministic makespan $D_q(s_e) \leq D_q^* \times (1 + i/100)$ are simulated and the one with the lowest probabilistic makespan is returned. If an optimal deterministic solution cannot be found within the CPU time limit, the best deterministic solution found is simulated and returned (i.e., only one simulation is done). Algorithm 3 presents the pseudocode.

The algorithm is theoretically complete. When i is large enough that the cost bound is greater than the deterministic makespan of *all* activity permutations, they will all be simulated. However, i would have to grow unreasonably large and therefore we treat this algorithm as, practically, incomplete. We refer to this algorithm as *B&B-I-BS* for *B*ranch-and-*B*ound-*I*terative-*B*est *S*olution.

B&B-I-BS(q):
Returns the solution with smallest probabilistic makespan

```
1  (s*, D_initial) ← findOptB&B(∞, q, t_0 − 1)
2  D* ← simulate(s*)
3  i ← 0
4  while not timed-out do
5      while termination criteria unmet do
6          (s, D_q) ← findNextB&B(
           D_initial × (1 + i/100), q, t_remaining)
7          D ← simulate(s)
8          if D < D* then
9              s* ← s; D* ← D
           end
       end
10     i ← i + 1
   end
11 return s*
```

Algorithm 3: B&B-I-BS

3.4 Local Search Filtering Algorithms

We also experiment with two local search filtering algorithms analogous to the constraint-based algorithms above except that tabu search is used in place of constraint-based search.

Tabu Timed Better Solution For a fixed q and for a fixed CPU time, $t_{initial}$, a solution with the lowest deterministic makespan, $D_{initial}$, possible is sought. Search is then restarted and whenever a solution, s, is found that has a deterministic makespan $D_q(s) \leq D_{initial}$, simulation is used to evaluate the probabilistic makespan. The solution with the lowest probabilistic makespan is returned. The pseudocode for Tabu-TBS is the same as that for B&B-TBS (Algorithm 2) with the following changes. The function findNextTabu(c, q, t) replaces findNextB&B(c, q, t); it produces a sequence of solutions whose deterministic makespan is less than c. The findBestTabu(c, q, t) function replaces findOptB&B(c, q, t); tabu search is run for up to t CPU seconds and the solution with the lowest deterministic makespan that is less than c is returned. We call this algorithm *Tabu-TBS* for *Tabu-T*imed *B*etter *S*olution.

Tabu Iterative Best Solution The core tabu search implementation does not necessarily use the entire CPU time. We can therefore create an iterative tabu-based solver for the probabilistic JSP similar to B&B-I-BS. In the first phase, using the overall time limit less one second, tabu search is used to find a good deterministic solution, based on a fixed q value. That solution is then simulated. Any remaining time is spent generating solutions with a deterministic makespan within a fixed percentage of the initial solution's deterministic makespan. As with B&B-I-BS, iterations are run with increasing i value starting with $i = 0$. In each iteration, solutions found by the tabu search whose deterministic makespan

are within $1 + i/100$ of the initial deterministic makespan are simulated. The solution with the lowest probabilistic makespan is returned. The algorithm is termed *Tabu-I-BS* for *Tabu-I*terative-*B*est *S*earch. The pseudocode for this algorithm is the same as that for B&B-I-BS (Algorithm 3) with the same substitutions that were made for Tabu-TBS.

3.5 Core Algorithm Details

The branch-and-bound algorithms described use texture-based heuristics [Beck and Fox, 2000] to decide on the the pair of activities to sequence and which sequence to try first. When constraint propagation is used (i.e., all B&B algorithms except B&B-N), we use temporal propagation, timetables [Le Pape *et al.*, 1994], edge-finder [Nuijten, 1994], and the balance constraint [Laborie, 2003].

The tabu search is the same one used by Beck & Wilson: the TSAB algorithm of [Nowicki and Smutnicki, 1996].

4 Empirical Investigations

Our empirical investigations aim to evaluate and investigate the performance of the algorithms. In particular, we are interested in algorithm scaling with problem size and uncertainty level and with the usefulness of representing uncertainty information using q values.

4.1 Experimental Details

We use four sets of probabilistic JSPs of size $\{4\times4, 6\times6, 10\times10, 20\times20\}$ with three uncertainty levels $u_j \in \{0.1, 0.5, 1\}$. A deterministic problem is generated using an existing generator [Watson *et al.*, 2002] with integer durations drawn uniformly from the interval [1, 99]. Three probabilistic instances are then produced by setting the mean durations μ_i to be the deterministic durations and by uniformly drawing the standard deviation σ_i from the interval $[0, u_j\mu_i]$. The distribution of each duration is approximately normal. For each problem size, we generate 10 deterministic problems which are transformed into 30 probabilistic instances.

Given the stochastic nature of the simulation and the tabu search algorithm, each algorithm is run 10 times on each problem instance with different random seeds. Each run has a time limit of 600 CPU seconds. Each Monte Carlo simulation uses 1000 trials. For B&B-TBS and Tabu-TBS, $t_{initial}$ is 60 CPU seconds.

For the heuristic techniques, a deterministic duration is assigned to each of the activities based on the q value. We experiment with the same four q values used by Beck & Wilson as displayed in Table 1. The B&B-DQ-L algorithm requires an initial value of q, q_0, and a decrement, q_{dec}. For all except the largest problems, we used $q_0 = 1.25$ and $q_{dec} = 0.05$. Preliminary experiments with the 20×20 problems indicated that $q_0 = 1.25$ resulted in long runs with no solutions due to a large search space with few if any solutions. To avoid such runs, we used $q_0 = 0.9$, for which solutions could be found.

The probabilistic makespans are based on confidence level $p = 0.95$. Our primary evaluation criterion is the mean normalized probabilistic makespan (*MNPM*) that each algorithm achieves: $\mathit{MNPM}(a, L) = \frac{1}{|L|}\sum_{l\in L}\frac{D(a,l)}{D_{lb}(l)}$. L is a set of problem instances, $D(a, l)$ is the mean probabilistic makespan found by algorithm a on l over 10 runs, $D_{lb}(l)$ is the lower bound on the probabilistic makespan for l. For all problems except 20×20, the D_{lb} is found by solving the deterministic problems to optimality using the q_1 durations (see Section 3 of [Beck and Wilson, 2004]). For the 20×20 problems, the optimal solutions could not be found and so the D_{lb} values represent the best deterministic solutions found. These values, therefore, are not a true lower bound.

q_0	q_1	q_2	q_3
0	$\frac{B}{\sqrt{2n}}$	$\frac{q_1+q_3}{2}$	$\frac{B}{\sqrt{n}}\frac{\sqrt{\text{Average}_{A_i\in\pi}\sigma_i^2}}{\text{Average}_{A_i\in\pi}\sigma_i}$

Table 1: The q-values used, following Beck & Wilson. n is the number of jobs and $B = 1.645$.

The hardware used for the experiments is a 1.8GHz Pentium 4 with 512 M of main memory running Linux RedHat 9. All algorithms were implemented using ILOG Scheduler 5.3.

4.2 Results

Table 2 presents the results of our experiment. Each cell is the mean over 10 independent runs of 10 problems. The observed mean standard deviations for each cell are small: the maximum over all cells is less than 10.3% of the corresponding mean makespan, while the mean over all cells is less than 0.8%. We did not observe a large difference among the q values provided $q > 0$ and therefore present the results for q_2.

Problem Size	Unc. Level	Algorithms					
		B&B Complete		B&B Heuristic		tabu	
		N	DQ-L	TBS	I-BS	TBS	I-BS
4×4	0.1	1.027*	**1.023***	1.026	1.026	1.027	**1.023**
	0.5	1.060*	1.049*	1.064	1.059	1.061	**1.046**
	1	1.151*	1.129	1.154	1.149	1.150	**1.128**
6×6	0.1	1.034	**1.021**	1.022	1.022	1.025	1.023
	0.5	1.113	**1.073**	1.083	1.077	1.089	1.074
	1	1.226	1.170	1.178	1.174	1.182	**1.168**
10×10	0.1	1.185	1.028	**1.024**	**1.024**	1.035	1.028
	0.5	1.241	1.115	**1.101**	**1.101**	1.121	1.112
	1	1.346	1.234	**1.215**	**1.215**	1.244	1.223
20×20†	0.1	1.256	1.142	1.077	1.071	1.029	**1.027**
	0.5	1.326	1.233	1.177	1.181	**1.136**	1.137
	1	1.482	1.388	1.334	1.338	**1.297**	1.307

Table 2: The mean normalized probabilistic makespans for each algorithm. '*' indicates a set of runs for which we have, with high confidence, found close-to-optimal makespans. '†' indicates problem sets for which normalization was done with approximate lower bounds. The lowest *MNPM* found for each problem set are shown in bold.

For the 4×4 problems, a number of algorithms find lower mean probabilistic makespans than B&B-N despite the fact that B&B-N terminates before the limit on CPU time and therefore finds approximately optimal solutions. This is an artifact resulting from algorithms that simulate the same solution multiple times. In B&B-N, a particular solution is only simulated once. In other algorithms, the same solution may be simulated multiple times leading to a bias to lower probabilistic makespan values. Based on the theoretic model of Beck & Wilson, these values still correspond to approximately optimal solutions.

4.3 Analysis

Overall the empirical results agree with those presented by Beck & Wilson. Here, we will focus on investigations of B&B-DQ-L and of the behavior of the heuristic algorithms.

Branch-and-Bound Descending-Q Leaves

B&B-DQ-L out-performs B&B-N across all problem sets even when it is not able to reach $q = 0$. As an indication of the number of iterations, the mean minimum values q for each problem size are: $4 \times 4 : 0.007$, $6 \times 6 : 0.56$, $10 \times 10 : 1.00$, $20 \times 20 : 0.9$.

For B&B-DQ-L, the deterministic durations defined by the q value serve to guide and prune the search for each iteration and, therefore, as with the heuristic algorithms (see below), the search is heuristically guided to the extent that solutions with low deterministic makespans also have low probabilistic makespans. However, the characteristics of the solutions found by the search are unclear. Ideally, we would like the search with high q to find solutions with very good probabilistic makespans both because we wish to find good solutions quickly and because the probabilistic makespan values are used to prune subsequent search. Analysis indicates that the first solution found by B&B-DQ-L is consistently better than the first solution found by B&B-N and for the larger problems (10×10 and 20×20) the relative increase in solution quality over subsequent iterations is greater for B&B-DQ-L.

Heuristic Algorithms

To investigate the performance of the heuristic algorithms, we look at the performance differences coming from using the q values, and the extent to which the performance of the heuristic algorithms depends on two factors: their ability to find good deterministic solutions and the underlying correlation between deterministic and probabilistic makespans in the experimental problems.

The Effect of the q Values Using a non-zero q value almost always results in better performance. This appears to be particularly true when either the uncertainty or the problem size increases. Table 3 shows the difference between mean normalized probabilistic makespan when using q_0 and q_2. On a few problem sets, negative values indicate that the runs with q_0 deliver a better mean solution. However, by far the majority of the values are greater than 0 and their magnitude increases with both uncertainty and problem size, indicating increasingly good relative performance on the q_2 problems. We show below that one explanation for this performance is that problem instances with $q > 0$ have a greater correlation between deterministic and probabilistic makespans.

Finding Good Deterministic Makespans We hypothesize that the performance of the heuristic techniques (and B&B-DQ-L) is partially dependent upon the ability to find solutions with low deterministic makespans. We looked at two measures of the performance of B&B-I-BS and Tabu-I-BS: the quality of the best deterministic solutions found and the number of iterations. Because B&B-I-BS performs systematic search of all deterministic solutions with makespan below a given threshold, we hypothesize that on problem sets where it outperforms Tabu-I-BS, it will also have found better deterministic solutions and/or be able to systematically search solutions up to a higher threshold. In contrast, for problem sets where Tabu-I-BS outperforms B&B-I-BS, it will be due to finding better deterministic solutions.

Problem Size	Unc. Level	B&B		Tabu	
		TBS	I-BS	TBS	I-BS
4×4	0.1	0.001	0	0	0
	0.5	0	0.001	0.001	0.001
	1	0.02	-0.001	0.021	0.017
6×6	0.1	0.002	0	0.001	0.001
	0.5	-0.002	0	0.005	0.004
	1	0.027	0.001	0.041	0.009
10×10	0.1	0	0	-0.001	0.002
	0.5	0.004	0.004	0.006	0.002
	1	0.015	0.016	0.017	0.009
20×20†	0.1	0.009	0.010	0.011	0.015
	0.5	0.010	0.005	0.012	0.028
	1	0.025	0.030	0.029	0.032

Table 3: The difference between the mean normalized probabilistic makespans for runs using q_0 and using q_2. The higher the value, the more the q_2 runs out-performed the q_0 runs.

Table 4 presents data using $q = q_2$. The mean normalized deterministic makespan (*MNDM*) is the calculated as follows: $\mathit{MNDM}(a, L) = \frac{1}{|L|} \sum_{l \in L} \frac{D_q(a,l)}{D_{q,min}(l, B\&B-I-BS)}$. The notation is as above with the addition of $D_{q,min}(l, \mathit{B\&B\text{-}I\text{-}BS})$, the lowest deterministic makespan found by the B&B-I-BS algorithm over all runs on problem l. MNDM, therefore, provides a relative measure of the mean deterministic makespans from the two algorithms: the higher the value, the worse is the average makespan found relative to B&B-I-BS. The second measure (*Iters*) is the mean number of iterations performed by each algorithm.

Problem Size	Unc. Level	B&B-I-BS		Tabu-I-BS	
		MNDM	Iters	MNDM	Iters
4×4	0.1	1.000	100	1.000	76
	0.5	1.000	100	1.000	78
	1	1.000	100	1.007	83
6×6	0.1	1.000	10	1.000	24
	0.5	1.000	10	1.001	31
	1	1.000	11	1.000	21
10×10	0.1	1.000	1	1.002	5
	0.5	1.000	1	1.004	5
	1	1.000	1	1.004	5
20×20	0.1	1.045	0	1.002	0
	0.5	1.041	0	0.998	0
	1	1.037	0	1.002	0

Table 4: The mean normalized deterministic makespan (MNDM) and number of iterations (Iters) for B&B-I-BS and Tabu-I-BS.

Table 4 is consistent with our hypotheses. For those problems sets with large performance differences (i.e., 10×10 and 20×20) in probabilistic makespan, the better performing algorithm does find better deterministic makespans.

The Correlation Between Deterministic and Probabilistic Makespan The explanatory power of an algorithm's ability to find good deterministic makespans is, by itself, insufficient unless there is a correlation between the good deterministic and good probabilistic makespans. It is reasonable to expect that the level of uncertainty has an impact on any correlation: at low uncertainty, the variations in duration are small, so we can expect the probabilistic makespan to be relatively close to the deterministic makespan. When the uncertainty level is high, the distribution of probabilistic makespans for a single solution will be wider, resulting in a lower correlation. We hypothesize that this impact of uncertainty level contributes to the observed performance degradation (see Table 2) of the heuristic techniques with higher uncertainty levels.

We generated 100 new 10×10 deterministic JSP problem instances with the same generator and parameters used above. The standard deviation for the duration of each activity in the 100 instances were generated independently for each of five uncertainty levels $u_j \in \{0.1, 0.5, 1, 2, 3\}$ resulting in a total of 500 problem instances. For each instance and each of the four q values above, we randomly generated and simulated 100 deterministic solutions. Using R [R Development Core Team, 2004], we measured the correlation coefficient, r, for each problem set.

Unc. Level	q_0	q_1	q_2	q_3
0.1	0.9990	0.9996	0.9996	0.9995
0.5	0.9767	0.9912	0.9917	0.9909
1	0.9176	0.9740	0.9751	0.9736
2	0.8240	0.9451	0.9507	0.9517
3	0.7381	0.9362	0.9418	0.9423

Table 5: The correlation coefficient comparing deterministic and probabilistic makespans for a set of 10×10 probabilistic JSPs. Each cell represents the correlation coefficient for 10000 deterministic, probabilistic pairs.

Table 5 supports our hypothesis. As the uncertainty level increases, the correlation between the deterministic makespan and corresponding probabilistic makespan lessens. The strength of the correlation is somewhat surprising: even for the highest uncertainty level, r is more than 0.94 for q_2 and q_3. This suggests that the heuristic algorithms can be successfully applied to problem with higher uncertainty levels provided the appropriate q value can be found.

The correlation results in Table 5 also provide an explanation for the effect of the q values: they increase the correlation between deterministic and probabilistic makespans.

5 Conclusion

In this paper, we presented and conducted empirical analysis of a number of algorithms to proactively solve the job shop scheduling problem with probabilistic durations. Our empirical studies have demonstrated that a novel algorithm, B&B-DQ-L, based on iteratively reducing a parameter that determines the validity of the lower bound results in equal performance on small problems and much better performance on larger problems when compared to the previous branch-and-bound technique. However, highest performance was observed in heuristic algorithms based on using the deterministic makespan to filter the solutions which are to be simulated. It was argued, using detailed analysis of the experimental data and results from an ancillary experiment, that the performance of these algorithms is affected by their ability to find good deterministic makespans and the correlation in the problems between the quality of deterministic and probabilistic makespans. The correlation data on problems with higher uncertainty suggest that such algorithms may scale well to such problems. Central to the success of the heuristic algorithms was the use of a q value that governs the extent to which uncertainty (i.e., the variance) was represented in the durations of activities in deterministic problems. It was shown that such an incorporation of uncertainty data leads to a stronger correlation between deterministic and probabilistic makespans, and suggests a corresponding ability to find better probabilistic makespans.

References

[Beck and Fox, 2000] J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.

[Beck and Wilson, 2004] J.C. Beck and N. Wilson. Job shop scheduling with probabilistic durations. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI04)*, pages 652–656, 2004.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[Laborie, 2003] P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143:151–188, January 2003.

[Le Pape *et al.*, 1994] C. Le Pape, P. Couronné, D. Vergamini, and V. Gosselin. Time-versus-capacity compromises in project scheduling. In *Proceedings of the Thirteenth Workshop of the UK Planning Special Interest Group*, 1994.

[Nowicki and Smutnicki, 1996] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[Nuijten, 1994] W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.

[R Development Core Team, 2004] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. ISBN 3-900051-07-0.

[Watson *et al.*, 2002] J.-P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14(1), 2002.